ERDC MSRC/PET TR/00-34

Tools For Understanding Program Performance

by

John Mellor-Crummey

31 July 2000

Work funded by the DoD High Performance Computing Modernization Program ERDC Major Shared Resource Center through

Programming Environment and Training (PET)

Supported by Contract Number: DAHC94-96-C0002
CSC Nichols

Tools for Understanding Program Performance

ERDC MSRC PET Project Report
July 31, 2000

Dr. John Mellor-Crummey, Rice University

Introduction

Over the past two decades, the peak performance of modern computers has increased dramatically through a collection of innovations in computer architecture. However, the increased potential for performance has come at the expense of architectural complexity. Achieving a substantial fraction of peak performance on modern systems requires exploiting the system's capabilities effectively.

In today's computers there is a substantial difference in speed between processors and system memory. To bridge this gap, commodity computer systems use multi-level memory hierarchies. Typically, such memory hierarchies contain two or more layers of cache and a translation look-aside buffer. It is impossible to achieve good performance on computer systems that use memory hierarchies unless most of a program's data references are satisfied either from the processor's registers or the top level of the memory hierarchy. Otherwise, the processor will spend much of its time idle waiting for data to arrive. To ensure that most of the program's data references are satisfied at the top levels of the memory hierarchy, the program must exploit spatial and temporal reuse of data.

Besides making efficient use of a memory hierarchy, to achieve high performance a program must also effectively exploit all of a system's capabilities. For example, a scientific program must address a range of issues from ensuring that instruction pipelines are full to issuing memory references in bunches so that multiple misses are outstanding when the processor stalls waiting for memory.

Understanding the details of such complex issues is extremely difficult for application developers. Effective tools to support execution analysis and program tuning are essential. To help DoD users identify performance bottlenecks and to tune programs to exploit modern computer systems effectively, we constructed two tools. The focus of our effort has been to develop tools that are easy to use rather than inventing new measures or new ways to collect them.

Our focused effort originally aimed at developing MHSIM - a simulator and program instrumentation tool for pinpointing the causes of poor memory hierarchy utilization in Fortran 77 programs, including those that use Fortran 90 array operations as shorthand for loops. Dr. Jane Smith of ERDC MSRC supplied us with STWAVE as an example of the type of Fortran applications for which ERDC users would like detailed performance feedback to support program tuning. After examining STWAVE, it became apparent that our tool for instrumenting Fortran programs for use with the MHSIM simulator would need much more extensive support for Fortran 90 language constructs than had been originally proposed as part of this focused effort. To address this problem, we designed and developed a second tool, HPCView, a language-independent tool that uses hardware performance counters rather than simulation as the basis for collecting performance information. While hardware performance counters can be used to accurately measure the memory hierarchy utilization by identifying the statements that cause cache misses, they can also be used to diagnose a wide range of performance problems by pinpointing where the program spends its cycles and what operations are being performed there. We designed HPCView to correlate different types of program metrics (e.g., cycles, floating point operations, and cache misses) with each other and with the program source. The MHSIM and HPCView tools are complementary. HPCView can be used to identify *where* the key performance bottlenecks in a program are located. MHSIM supports detailed analysis of memory hierarchy behavior and can help understand *why* the memory hierarchy utilization of a program is poor. In the following sections, we briefly describe the MHSIM and HPCView tools.

Both tools produce hyper-linked HTML documents for exploration with Netscape Navigator. This user interface strategy has three key advantages. First, it eliminates the need to develop a custom user interface. Netscape Navigator provides a rich interface that includes the ability to search, scroll, and navigate hyperlinks within documents. Also, it provides users with what has become a familiar paradigm for navigating information.

MHSIM

For an application code to achieve high performance, it must exploit caches effectively. Most scientific codes in production use were developed for vector processors that had no caches. When porting such applications to machines with multiple layers of caches, it is difficult to understand the reasons for poor memory hierarchy utilization.

To address this problem we have developed MHSIM, an integrated instrumentation tool and simulator. MHSIM is designed to

- identify program references causing poor cache utilization,
- quantify cache conflicts, temporal reuse and spatial reuse, and
- correlate simulation results to references and loops in an application program.

The MHSIM simulator and associated instrumentation tool orchestrate a detailed simulation of multi-level memory hierarchies that can help identify the causes responsible for poor memory hierarchy utilization and help users achieve a higher fraction of peak performance. The impact and DoD relevance of this tool is to help DoD users understand how to tune memory hierarchy performance of complex Fortran 77 and Fortran 90 application codes.

The MHSIM simulator and instrumentation tool for Fortran programs is available from http://www.cs.rice.edu/~dsystem/mhsim. The distribution contains source code for the simulator and Solaris and Irix binaries for the instrumentation tool. A portable source-code release of the instrumentation tool will soon be available as well as part of a distribution of the Rice Dsystem program analysis and compilation tools for Fortran. Documentation in the release includes instructions for building the simulator library, online help information about how to navigate the simulator results, and several directories of examples with Makefiles that demonstrate how to use the simulator to track memory hierarchy utilization.

Using the MHSIM simulator to understand the memory hierarchy utilization of a Fortran program involves a sequence of five steps. First, the *libmhsim* simulator library is configured using a configuration file that specifies parameters of the target system. Second, the Fortran source code under study is instrumented using *mhinst*, a source-to-source instrumentation tool. The instrumentation tool augments the program with calls to the libmhsim library routines to monitor data accesses and relate simulator results back to the source code. Third, the instrumented Fortran code is then compiled with any Fortran compiler and linked with the libmhsim library. Next, the compiled version of the instrumented program is executed normally on any platform. During execution, the calls added by the instrumenter will pass information about the program's data accesses to the simulator, which will track information about the program's memory hierarchy utilization separately for each source-program reference. When the program finishes executing, the simulator correlates the simulation results with the source program and writes them out as a hyper-linked HTML database that forms the basis of a multi-pane user interface. Finally, a user loads the root page of the simulation results and interactively inspects them with Netscape Navigator. Each of these steps will be explained briefly below.

Configuring the simulator

Simulator configuration files specify a number of parameters about the memory hierarchy of the target system being simulated:

- number of levels of memory hierarchy,
- number of cache lines,
- line size,
- associativity,
- write-through or write back, and
- translation-lookaside-buffer configuration.

The format of the configuration files is documented by sample configurations provided in the MHSIM software release. The release contains configurations for several sample memory hierarchies including a fully associative cache, and configurations matching the characteristics of an SGI O2, and an SGI Origin 2000.

The configuration file is processed with a simulator configuration tool which instantiates a configured version of the routine that simulates memory access. The code to simulate an access to a particular cache is coded as a C++ template. The configuration tool specializes the cache_access template for each level of the memory hierarchy and passes the configuration constants for that level of the memory hierarchy as parameters to the template. The C++ template instantiator uses these constants to instantiate a customized version of the cache simulation routine for each level of the memory hierarchy. The resulting code has all of the constants for cache, line, and set sizes as compile-time constants, which gives the compiler information to optimize the object program more effectively.

Running the *mhinst* instrumentation tool

The mhinst program instrumentation tool processes Fortran programs and adds calls to the libmhsim library to record source-code mapping information for every array reference, loop, and procedure. We illustrate the instrumentation process with an example. Consider the following source program fragment:

```
              leak = 0.0
                  k = k0 - k2
                  do mi = 1, mmi
                   m = mi + mio
                  do j = 1, jt
                  do i = 1, it
                     phikb(i,j,mi) = phikbc(i,j,m)
                     leak = leak
     *                    + wtsi(m)*phikb(i,j,mi)*di(i)*dj(j)
                     face(i,j,k+k3,3) = face(i,j,k+k3,3)
     *                    + wtsi(m)*phikb(i,j,mi)
                  end do
                  end do
                  end do
                  leakage(5) = leakage(5) + leak
```

The instrumented version of the fragment is shown below.

```
 leak = 0.0
 k = k0 - k2
 call mhsim_enter_scope(mhsim_scope_11)
 do mi = 1, mmi
   m = mi + mio
   call mhsim_enter_scope(mhsim_scope_12)
   do j = 1, jt
     continue
     call mhsim_enter_scope(mhsim_scope_13)
     do i = 1, it
       call mhsim(MHSIM_RARRAY, 8, phikbc(i, j, m), mhsim_ref_14,
     *           MHSIM_WARRAY, 8, phikb(i, j, mi), mhsim_ref_15, MHSIM_NONE)
       phikb(i, j, mi) = phikbc(i, j, m)
       call mhsim(MHSIM_RARRAY, 8, dj(j), mhsim_ref_16, MHSIM_RARRAY, 8, di(i),
     *           mhsim_ref_17, MHSIM_RARRAY, 8, phikb(i, j, mi), mhsim_ref_18,
     *           MHSIM_RARRAY, 8, wtsi(m), mhsim_ref_19, MHSIM_NONE)
       leak = leak + wtsi(m) * phikb(i, j, mi) * di(i) * dj(j)
       call mhsim(MHSIM_RARRAY, 8, phikb(i, j, mi), mhsim_ref_20, MHSIM_RARRAY, 8,
     *           wtsi(m), mhsim_ref_21, MHSIM_RARRAY, 8, face(i, j, k + k3, 3),
     *           mhsim_ref_22, MHSIM_WARRAY, 8, face(i, j, k
     *          + k3, 3), mhsim_ref_23, MHSIM_NONE)
       face(i, j, k + k3, 3) = face(i, j, k + k3, 3) + wtsi(m) * phikb(i, j, mi)
     enddo
     call mhsim_exit_scope(mhsim_scope_13)
     continue
   enddo
   call mhsim_exit_scope(mhsim_scope_12)
   continue
 enddo
 call mhsim_exit_scope(mhsim_scope_11)
 call mhsim(MHSIM_RARRAY, 8, leakage(5), mhsim_ref_24, MHSIM_WARRAY, 8, leakage(5),
      *          mhsim_ref_25, MHSIM_NONE)
 leakage(5) = leakage(5) + leak
```

Loop entry and exits are instrumented with calls to mhsim_enter_scope and mhsim_exit_scope respectively. Each scope entry/exit operation is passed a handle that identifies the source location of the scope. These calls perform bookkeeping for constructing loop level summary statistics.

The instrumentation tool adds calls to mhsim before each array access to simulate the response of the memory hierarchy. Each simulator call is passed a type code indicating whether the operation is a READ or WRITE of an array, the number of bytes accessed, the address of the access, and the handle for the source reference corresponding to the access. Below we show initialization of a handle for a sample scope.

```
      call mhsim_init_scope(mhsim_scope_13, 'sweep.mhsim.src.f', 199,
     *                      '', MHSIM_LOOP_SCOPE, mhsim_scope_12)
```

This call creates a scope handle known as mhsim_scope_13 that corresponds to the loop on line 199 in file sweep.mhsim.src.f (a preprocessed version of the source file sweep.f generated by the instrumentation tool) which is enclosed in the loop identified by the handle mhsim_scope_12. Initialization of a handle for a reference is similar as shown below.

```
      call mhsim_init_ref(mhsim_ref_23, MHSIM_WARRAY, 'face(i,j,k+k3
     *                ,3)', 'sweep:face', mhsim_scope_13, 202, 23, 43, 8)
```

This call creates a reference handle known as mhsim_ref_23 for a WRITE access to the reference face(i,j,k+k3,3), an 8-byte element of the local variable face declared in routine sweep. The reference is on line 202 spanning character positions 23-43. The reference occurs inside the scope mhsim_scope_13. The simulator uses these handles to relate accesses to source program references.

Limitations of source-code-based instrumentation and simulation

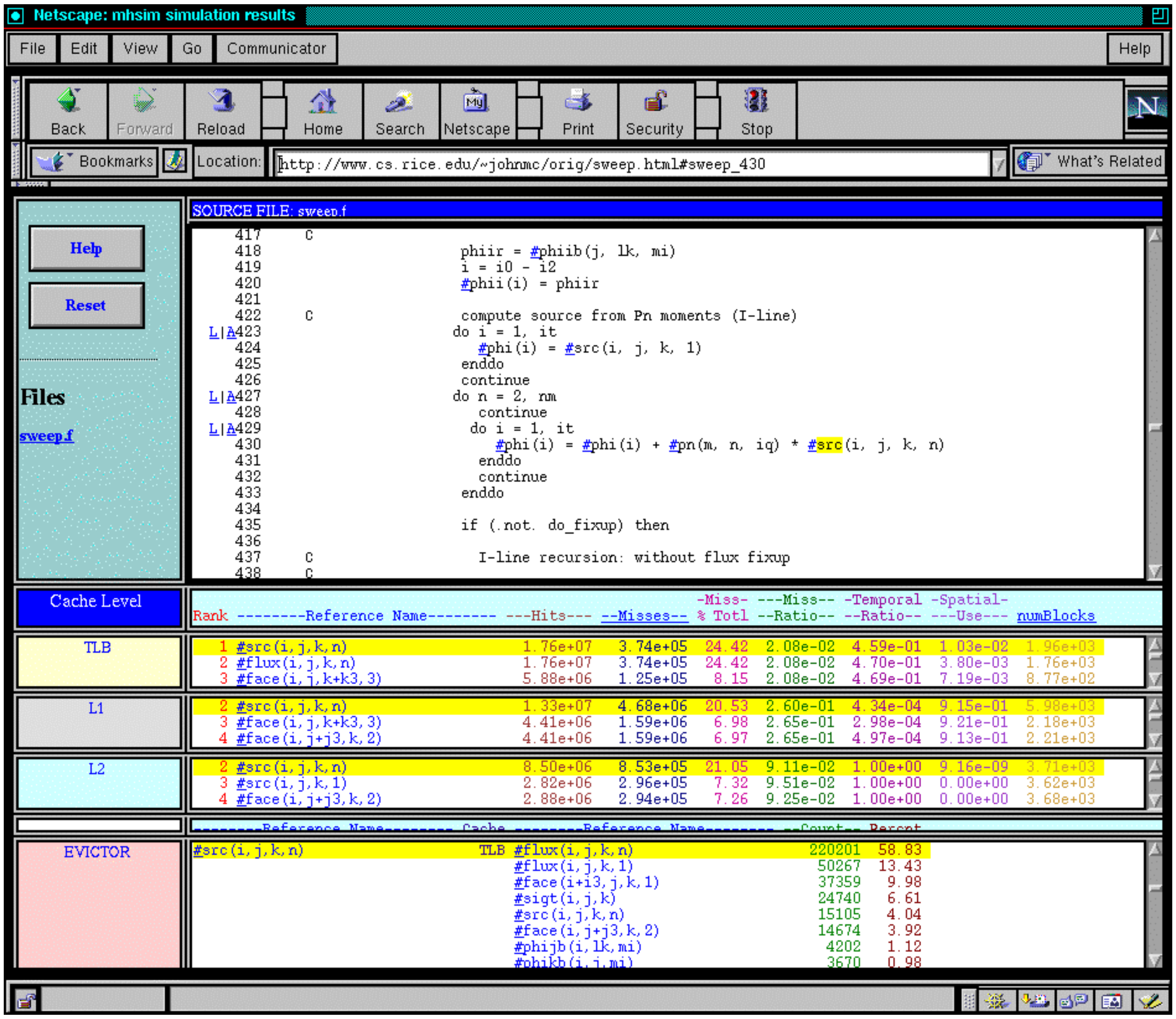
The MHSIM simulator uses the addresses passed to the instrumentation routines as the basis for its simulation. The declarations of handles that are inserted by the instrumentation tool perturb the addresses of the program variables. This perturbation affects the absolute position of the data elements which affects the cache conflicts noted by the simulator. Our experience is that the perturbation does not substantially affect the qualitative nature of the results, although in the worst case it could.

A second limitation of the source-code instrumentation strategy used by mhinst is that the data accesses specified in the source program will be simulated in their canonical execution order. Any compiler-based loop transformations that might be performed by the back-end compiler are not accounted for.

Exploring MHSIM simulation results

The MHSIM simulator records the results of its simulation in a collection of HTML files that can be browsed using Netscape Navigator version 4.0 or higher. The hyperlinked HTML files generated by MHSIM use a set of Javascript routines to provide an integrated user interface. The top level display of the simulator results is shown below.

The top left pane of the display shows a list of instrumented source files in the program. For this simulation, there was one file: sweep.f.



http://www.cs.rice.edu/~johnmc/orig/sweep.html#sweep_430

SOURCE FILE: sweep.f

```
     C
                    phiir = #phiib(j, lk, mi)
                    i = i0 - i2
                    #phii(i) = phiir

     C              compute source from Pn moments (I-line)
L|A                do i = 1, it
                      #phi(i) = #src(i, j, k, 1)
                    enddo
                    continue
L|A                do n = 2, nm
                      continue
                     do i = 1, it
                        #phi(i) = #phi(i) + #pn(m, n, iq) * #src(i, j, k, n)
                      enddo
                      continue
                    enddo

                    if (.not. do_fixup) then

     C                I-line recursion: without flux fixup
     C
```

```
                                                -Miss- ---Miss-- -Temporal -Spatial-
Rank --------Reference Name-------- ---Hits--- --Misses-- % Totl --Ratio-- --Ratio-- ---Use--- numBlocks
   1 #src(i,j,k,n)                  1.76e+07   3.74e+05  24.42  2.08e-02  4.59e-01  1.03e-02  1.96e+03
   2 #flux(i,j,k,n)                 1.76e+07   3.74e+05  24.42  2.08e-02  4.70e-01  3.80e-03  1.76e+03
   3 #face(i,j,k+k3,3)              5.88e+06   1.25e+05   8.15  2.08e-02  4.69e-01  7.19e-03  8.77e+02

   2 #src(i,j,k,n)                  1.33e+07   4.68e+06  20.53  2.60e-01  4.34e-04  9.15e-01  5.98e+03
   3 #face(i,j,k+k3,3)              4.41e+06   1.59e+06   6.98  2.65e-01  2.98e-04  9.21e-01  2.18e+03
   4 #face(i,j+j3,k,2)              4.41e+06   1.59e+06   6.97  2.65e-01  4.97e-04  9.13e-01  2.21e+03

   2 #src(i,j,k,n)                  8.50e+06   8.53e+05  21.05  9.11e-02  1.00e+00  9.16e-09  3.71e+03
   3 #src(i,j,k,1)                  2.82e+06   2.96e+05   7.32  9.51e-02  1.00e+00  0.00e+00  3.62e+03
   4 #face(i,j+j3,k,2)              2.88e+06   2.94e+05   7.26  9.25e-02  1.00e+00  0.00e+00  3.68e+03
```

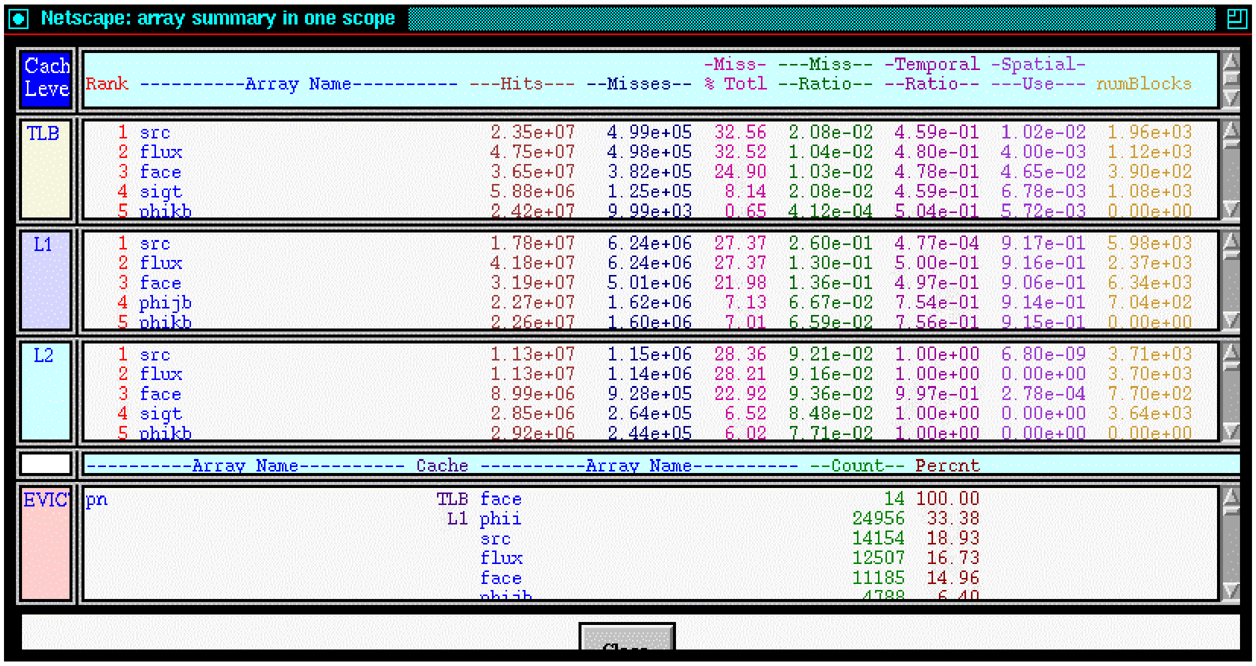
```
--------Reference Name-------- Cache --------Reference Name-------- --Count-- Percent
#src(i,j,k,n)                   TLB #flux(i,j,k,n)                    220201  58.83
                                    #flux(i,j,k,1)                     50267  13.43
                                    #face(i+i3,j,k,1)                  37359   9.98
                                    #sigt(i,j,k)                       24740   6.61
                                    #src(i,j,k,n)                      15105   4.04
                                    #face(i,j+j3,k,2)                  14674   3.92
                                    #phijb(i,lk,mi)                     4202   1.12
                                    #phikb(i,j,mi)                      3670   0.98
```

The upper right pane displays the source code of sweep.f, annotated with hyperlinks that enable a user to access information available for a reference or loop with a single click. Clicking on a # hyperlink preceding a reference will autoscroll each of the panes below to display the simulation results associated with that reference. Next to each scope are two hyperlinks. The '**L**' hyperlink will cause loop summary information to be displayed in the panes below rather than the reference-level information shown in the figure. The '**A**' hyperlink will display loop-level summary information for each array referenced in the loop.

The next three panes show simulation results collected for the target memory hierarchy, which in this case consists of a TLB, primary and secondary cache. Each level of the cache shows the name of the associated source code reference. Clicking on the hyperlink preceding any reference in these panes will cause the source pane to navigate to the appropriate line in the appropriate source file and all other panes of simulation results to auto-scroll to display the information associated with this reference. For each reference, the panes for each memory hierarchy level show the number of hits and misses, the percentage of total misses this represents, the miss ratio for this reference, the fraction of the data reuse for this reference attributed to temporal locality (*i.e.*, number of temporal hits / number of total hits) and spatial locality (*i.e.*, used bytes / (line size * number of evictions)), and how many blocks the reference occupied at this memory hierarchy level. The sample display included shows that the source program reference src(i,j,k,n) on line 430 accounts for over 20% of the misses at each of the levels in the memory hierarchy.

The bottom pane in the top-level window shows evictor information. For a particular source code reference, the evictor pane shows which source-code references faulted in cache lines that caused a cache line containing this reference to be evicted from cache. The percentage of the evictions caused by each source code reference is shown. Clicking on the hyperlink for an evicting reference will auto-navigate the source and memory hierarchy panes to show the information available for the evicting reference.

Below, we show the array-level summary information associated with a particular scope (loop or procedure) that can be brought up in a separate window by clicking on the '**L**' associated with a scope. This summary pane displays the same types of information reported for references, but summarizes all references to each array (and its evictors) within the scope.



Cach Leve	Rank	----------Array Name----------	---Hits---	--Misses--	-Miss- % Totl	---Miss-- --Ratio--	-Temporal --Ratio--	-Spatial- ---Use---	numBlocks
TLB	1	src	2.35e+07	4.99e+05	32.56	2.08e-02	4.59e-01	1.02e-02	1.96e+03
	2	flux	4.75e+07	4.98e+05	32.52	1.04e-02	4.80e-01	4.00e-03	1.12e+03
	3	face	3.65e+07	3.82e+05	24.90	1.03e-02	4.78e-01	4.65e-02	3.90e+02
	4	sigt	5.88e+06	1.25e+05	8.14	2.08e-02	4.59e-01	6.78e-03	1.08e+03
	5	phikb	2.42e+07	9.99e+03	0.65	4.12e-04	5.04e-01	5.72e-03	0.00e+00
L1	1	src	1.78e+07	6.24e+06	27.37	2.60e-01	4.77e-04	9.17e-01	5.98e+03
	2	flux	4.18e+07	6.24e+06	27.37	1.30e-01	5.00e-01	9.16e-01	2.37e+03
	3	face	3.19e+07	5.01e+06	21.98	1.36e-01	4.97e-01	9.06e-01	6.34e+03
	4	phijb	2.27e+07	1.62e+06	7.13	6.67e-02	7.54e-01	9.14e-01	7.04e+02
	5	phikb	2.26e+07	1.60e+06	7.01	6.59e-02	7.56e-01	9.15e-01	0.00e+00
L2	1	src	1.13e+07	1.15e+06	28.36	9.21e-02	1.00e+00	6.80e-09	3.71e+03
	2	flux	1.13e+07	1.14e+06	28.21	9.16e-02	1.00e+00	0.00e+00	3.70e+03
	3	face	8.99e+06	9.28e+05	22.92	9.36e-02	9.97e-01	2.78e-04	7.70e+02
	4	sigt	2.85e+06	2.64e+05	6.52	8.48e-02	1.00e+00	0.00e+00	3.64e+03
	5	phikb	2.92e+06	2.44e+05	6.02	7.71e-02	1.00e+00	0.00e+00	0.00e+00

	----------Array Name----------	Cache	----------Array Name----------	--Count--	Percnt
EVIC	pn	TLB	face	14	100.00
		L1	phii	24956	33.38
			src	14154	18.93
			flux	12507	16.73
			face	11185	14.96
			phijb	4788	6.40

Examining the summary information presented in the array pane for a costly procedure, or the top-level program scope can identify what array or arrays result in the most misses in the memory hierarchy. These arrays are the appropriate focus for improving a program's overall performance.

HPCView

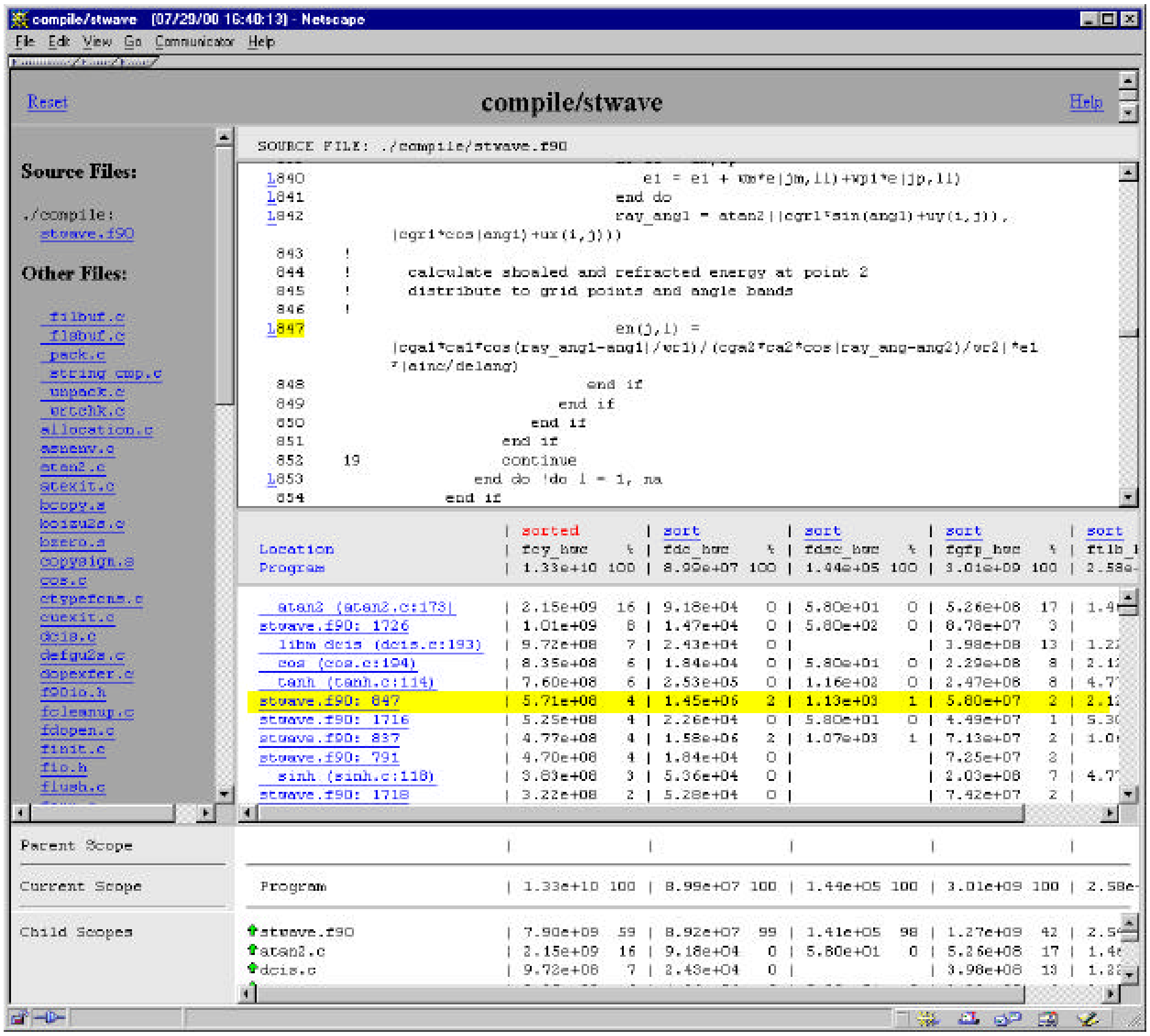
Interpreting the results of performance experiments is a key problem for application developers. Analysis often requires comparison and analysis of data from multiple experiments and sources. Interesting raw performance metrics for a program include dynamic measures such as the number of memory accesses or floating point operations performed by particular statements or the number of cycles spent executing these statements. Such measurements can help a user identify points in the program where significant effort is expended. Knowing where most of the effort is expended in a program helps a user understand where any improvements through program tuning are likely to yield the greatest benefits.

To facilitate performance analysis and program tuning, we developed HPCView, a tool for analyzing a program's performance data and correlating it with the program source. We designed the HPCView tool to present multiple performance metrics for an application program in a convenient form that can be easily understood. Performance data manipulated by HPCView can come from any source, as long as it is provided to the tool in the proper input format. To date, the principal source of input data for HPCView has been program counter (PC) histograms collected by sampling the program counter at periodic intervals, which are delineated by overflow of a hardware performance counter being used to count occurrences of a particular type of event (e.g. primary cache misses). These PC histograms are then interpreted using a program's symbol table information to correlate the measurements with the program source.

A distribution of the HPCView tool is available at http://www.cs.rice.edu/~dsystem/hpcview. Development of the HPCView tool is continuing under DOE support and future releases of the tool will be made available to the DoD modernization effort. The HPCView distribution contains source code for the tool. Documentation in the release includes several directories of examples with Makefiles that demonstrate how to use the tool to correlate multiple sources of performance data with an application program, and online help information about how to explore the program's performance results. Provided as part of the release is a script that converts performance profiles created by using SGI's prof utility to create line-level profiles of performance measurements collected using SGI's ssrun utility (which uses hardware performance counters to collect statistical profiles of particular types of program events).

The HPCView user interface

Like MHSIM, HPCView correlates performance data with an application program's source code and produces a hyper-linked database of HTML documents that forms the basis of a multi-pane user interface. A program performance database generated by HPCView can be interactively explored using Netscape Navigator version 4.0 or greater. The hyper-linked HTML files generated by HPCView use a set of Javascript routines to provide an integrated user interface. The top level display of the simulator results is shown below.



compile/stwave (07/29/00 16:40:13) - Netscape

compile/stwave

Source Files:

./compile:
stwave.f90

Other Files:

filbuf.c
flsbuf.c
pack.c
string_cmp.c
unpack.c
wrtchk.c
allocation.c
asmenv.c
atan2.c
atexit.c
bcopy.s
bcszu2s.c
bzero.s
copysign.s
cos.c
ctypefcns.c
cuexit.c
dcis.c
defgu2s.c
dopexfer.c
f90io.h
fcleanup.c
fdopen.c
finit.c
fio.h
flush.c

```
SOURCE FILE: ./compile/stwave.f90
1840                          e1 = e1 + wm*e(jm,ll)+wp1*e(jp,ll)
1841                       end do
1842                       ray_ang1 = atan2((cgr1*sin(ang1)+uy(i,j)),
        (cgr1*cos(ang1)+ux(i,j)))
 843    !
 844    !     calculate shoaled and refracted energy at point 2
 845    !     distribute to grid points and angle bands
 846    !
1847                       en(j,l) =
        (cga1*ca1*cos(ray_ang1-ang1)/wr1)/(cga2*ca2*cos(ray_ang-ang2)/wr2)*e1
        *(ainc/delang)
 848                     end if
 849                   end if
 850                 end if
 851               end if
 852    19         continue
1853             end do !do l = 1, na
 854           end if
```

```
                              | sorted         | sort           | sort           | sort           | sort
Location                      | fcy_hwc      % | fdc_hwc      % | fdsc_hwc     % | fgfp_hwc     % | ftlb_
Program                       | 1.33e+10 100   | 8.99e+07 100   | 1.44e+05 100   | 3.01e+09 100   | 2.58e-

  atan2 (atan2.c:173)         | 2.15e+09  16   | 9.18e+04   0   | 5.80e+01   0   | 5.26e+08  17   | 1.4
stwave.f90: 1726              | 1.01e+09   8   | 1.47e+04   0   | 5.80e+02   0   | 8.78e+07   3   |
  libm dcis (dcis.c:193)      | 9.72e+08   7   | 2.43e+04   0   |                | 3.98e+08  13   | 1.2
  cos (cos.c:194)             | 8.35e+08   6   | 1.84e+04   0   | 5.80e+01   0   | 2.29e+08   8   | 2.1
  tanh (tanh.c:114)           | 7.60e+08   6   | 2.53e+05   0   | 1.16e+02   0   | 2.47e+08   8   | 4.7
stwave.f90: 847               | 5.71e+08   4   | 1.45e+06   2   | 1.13e+03   1   | 5.80e+07   2   | 2.1
stwave.f90: 1716              | 5.25e+08   4   | 2.26e+04   0   | 5.80e+01   0   | 4.49e+07   1   | 5.3
stwave.f90: 837               | 4.77e+08   4   | 1.58e+06   2   | 1.07e+03   1   | 7.13e+07   2   | 1.0
stwave.f90: 791               | 4.70e+08   4   | 1.84e+04   0   |                | 7.25e+07   2   |
  sinh (sinh.c:118)           | 3.83e+08   3   | 5.36e+04   0   |                | 2.03e+08   7   | 4.7
stwave.f90: 1718              | 3.22e+08   2   | 5.28e+04   0   |                | 7.42e+07   2   |
```

```
Parent Scope                  |                |                |                |                |

Current Scope   Program       | 1.33e+10 100   | 8.99e+07 100   | 1.44e+05 100   | 3.01e+09 100   | 2.58e-

Child Scopes    stwave.f90    | 7.90e+09  59   | 8.92e+07  99   | 1.41e+05  98   | 1.27e+09  42   | 2.5
                atan2.c       | 2.15e+09  16   | 9.18e+04   0   | 5.80e+01   0   | 5.26e+08  17   | 1.4
                dcis.c        | 9.72e+08   7   | 2.43e+04   0   |                | 3.98e+08  13   | 1.2
```

The interface contains several panes:

- *The top pane at the top displays the title of the performance database.* The title pane also includes two hyperlinks. The 'Help' link on the right displays this manual. The 'Reset' link on the left reloads all files. The Reset button is intended for use if the browser becomes confused and has difficulty navigating the documents in the performance database.

- *The files pane on the left contains the list of files for which HPCView generated performance information.* The list of all known source files is grouped according to the directories in which the files were found. Within a directory listing, files are listed in alphabetical order. The "Other Files" section of this pane contains a list of files for which the HPCView tool could not find source. Clicking on any file in the list in this pane will display the selected item in the Source File Pane described below. The source file pane on the upper right contains the currently 'selected' source file.

- *The source-file pane displays information about the currently selected source file.* Lines are numbered for easy reference. Lines for which performance information can be found in the performance and scope hierarchy tables are marked with a **L** (which stands for line number information) in front of their line number. Clicking the **L** highlights the chosen line number and navigates to the corresponding row in the performance table. If the scope table currently displays the selected line, it highlights it as well.

  For any file for which the HPCView tool possesses performance information but cannot locate the corresponding source file, HPCView generates a file synopsis. The synopsis contains three lines for each of the file's procedures, which are generated according to the following scheme:

  ```
  L<line>    <subroutine name>
             ...
   <line>    end <subroutine name>
  ```

  For files for which no source code is available, HPCView collapses performance data for procedures within. HPCView uses the lowest source line number within a binary procedure as the line for which it reports the summary information for the procedure.

- *A table of line-level performance metrics is displayed in two panes.* One contains headings for each of the columns of performance metrics. The other contains a row for each source line that has at least one value for one of the metrics shown.

  When HPCView generates the performance table it uses a configuration file to determine what files contain the data for the performance metrics and what names the user wants to refer to the metrics by. HPCView uses the display names as column headers and the metric values to fill in the performance table.

  Each table row starts with a location field, which contains the file name and source line number. The following row elements contain the metric values found in the profiles. Row elements are empty if the metric's profile did not contain any information for that source line. Since metrics displayed by HPCView for a program are frequently generated by sampling directed by hardware performance counters, it can happen that some source lines have samples for one hardware metric, but not for another.

  In most cases table rows correspond to executable statements in a known source file. If HPCView can not find the source file for a procedure whose line is mentioned in a profile, it displays the metric summaries for this procedure in the performance table. Metrics are summarized by adding metric values of all lines in a procedure.

  The interface highlights a row if you click on the row's location field. It also shows and highlights the selected line in the source file pane. In some cases this means that a new source file is loaded. In other cases the currently shown source file scrolls the highlighted line into view.

The performance table is sorted according to the performance metric whose column header is shown in red. Clicking on the 'sort' link of the metric's column header will re-sort the data for both the performance table and the scope hierarchy display described below.

- The bottom of the window contains a hierarchical display of performance information for program scopes. This display presents inclusive performance statistics for the program, source files, procedures, and source lines. The three-part display shows inclusive performance measurements for the current scope, its parent scope, and its child scopes. The information displayed for each scope is in the same format as the performance table. Each row starts with a location field and is followed by values for each of the performance metrics.

  When an HPCView performance database is first opened with a browser, the scope hierarchy table display shows the program selected as the current scope with the program's files as the child scopes. To examine the performance information for a particular file, click on the arrow in front of its name and the scope panes will navigate to display the selected file the “Current Scope”, its procedures as ”Child Scopes”, and the program information in its “Parent Scope” row. In general clicking on the up- or down-arrows at the beginning of a row makes that scope the “Current Scope” and shows its parent and child scopes in the other lines in the display.

  The child scopes shown in the scope hierarchy display are sorted according to the performance metric whose column header in the performance table is shown in red. The data in the scope hierarchy display and the performance table can be sorted by another metric by clicking on the 'sort' link of the metric's column header.

Navigation of the information in a performance database is initiated by selecting a hyperlink for a line in any part of the interface. At any time you may select a source line in the interface. You do so by either selecting the L (which stands for line number information) in front of a line number in the source file pane or the location field at the beginning of a row in the performance table. The selected line is highlighted by the interface and the performance table scrolls so that metrics for the selected line are visible. If the selected line is one of the 'Child Scopes' in the scopes hierarchy display, that display will also scroll to make the selected line visible. If you now select another line the previously selected line is changed from highlighted in the source file pane, the performance table, and the scopes hierarchy display.

By selecting different lines from a particular code section you can investigate that section's performance. By selecting the location fields of rows with high performance metric values you can quickly find the source code that these performance values relate to.

Browser configuration

HPCView databases currently require using Netscape Navigator 4.0 or higher. Configuration information below assumes that you are using Navigator.

The top level window is a set of frames. Their relative sizes may be adjusted by clicking the middle button on a frame border and dragging it to increase/decrease frame sizes.

Depending on your Navigator window size you may want to adjust Navigator's settings so that text will be displayed in a bigger or smaller font. To do so choose "Preferences" under the "Edit" menu. In the Preferences Dialog you need to choose "Fonts" and change the font and/or size of both the Variable and Fixed Width fonts. Make sure "Use document-specified fonts, including Dynamic Fonts" is selected and click "OK".

Troubleshooting

HPCView databases currently require using Netscape Navigator 4.0 or higher**.** The performance databases are not yet compatible with Internet Explorer.

We recommend waiting until all files are completely loaded before clicking any links. Clicking on links before all data is loaded in any pane may interrupt loading and result in an incomplete setup. If this causes problems try to click the Reset button to reload files from scratch. If this does not help either clear Navigator's caches: choose "Preferences" under the "Edit", click on "Cache" in the "Category" pane, and clear the disk and memory cache; then try to reload.

Ongoing HPCView development

Understanding what opportunities for tuning exist may require understanding synthetic performance metrics such as the instruction mix in a program's loop nests (loop balance) compared to the ideal instruction mix supported by the target architecture. HPCView has been designed to support computing such synthetic performance metrics and relating them back to the program source. Computing metrics such as loop balance requires identifying where loops exist in application programs. Currently, we are in the process of designing and developing a program analysis tool to provide information about the location of loops to HPCView by analyzing application binaries.

Acknowledgements

This work was supported in part by the Department of Defense High Performance Computing Modernization Program ERDC MSRC Programming Environment and Training under Contract DAHC94-96-C002 and in part by the Department of Energy's Accelerated Strategic Computing Initiative under research subcontract B347884.